Deployment de serviços de orquestração cloud

Humberto Vaz e Miguel Costa

26 de Junho de 2018

1 Contexto/Motivação

As palavras *DevOps* e *Cloud Computing* têm vindo a ser bastante reconhecidas na informática dos dias de hoje. E na medida em que já existem vários serviços e aplicações desenvolvidas que correm só e apenas na cloud tirando partido da elasticidade, load balancing, e outras vantagens do *Platform as a Service* (PaaS). Com o intuito de tirar partido da possível gestão automatizada em ambiente de *Cloud Computing*, faz sentido explorar as potencialidades e dificuldades que nos trazem os *orquestradores* Open-Source. Keywords — Cloud Computing, DevOps, Orquestrador, Open-Source

2 Introdução

A nossa jornada começou com uma stack tecnológica constituída por três camadas, onde a meio dela se encontra um Orquestrador. Esse Orquestrador trata-se do `Janus`, uma aplicação paga que funciona como ponte entre o Gerenciador de Deployments, Alien4Cloud (A4C), e no nosso caso, o *OpenStack* sendo que este é um software Open-Source que permite controlar grandes grupos de computadores, isto é, Servidores `Bare Metal`, Servidores físicos dedicado a um único utilizador (Bare Metal), podendo criar opcionalmente um ambiente multi-cloud. O objetivo da nossa jornada será experimentar deployments de orquestradores *Open-Source* na cloud, identificar e analisar pontos sensíveis como incompatibilidades ao nível das API's, versionamento dos componentes e sobreposição de funcionalodades para substituír o `Janus`.

Após uma investigação mais detalhada, conseguiu-se perceber que as ferramentas utilizadas partilham funcionalidades. Isto é, a mesma ferramenta pode fazer o papel de mais do que uma camada. Como exemplo disso, o OpenStack, para além de IaaS pode funcionar também como Orquestrador.

2.1 Objetivos

Este trabalho, pretende demonstrar a utilização de uma stack tecnológica para fazer deploy de aplicações para num ambiente de Cloud Computing recorrendo

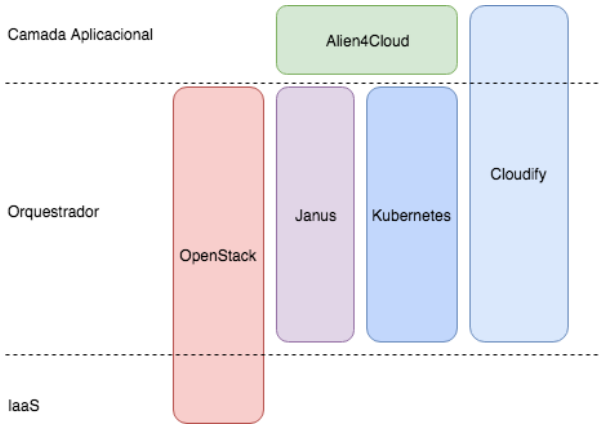

Figura 1: Ferramentas distribuídas pelas várias camadas

a `DevOps`, ou seja um serviço: *Service as a Platform* (SaaS). Este serviço oferece as seguintes vantagens:

- Lançar aplicações rapidamente
- Menos configurações
- Maior nível de abstração ao utilizador
- AutoScaling/Elasticidade

3 Revisão de literatura

3.0.1 Cloud

Acerca da cloud, trata-se de um grupo de computadores em rede virtualizados para fornecer serviços de forma fácil e eficiente. Com a adoção de técnicas de automatização é possível automatizar processos de aumento e diminuição de recursos, `elasticidade`, fazendo com que exista menor desperdício desses mesmos recursos, sendo beneficiário para ambas as partes, isto é, o utilizador e o fornecedor do serviço, havendo assim uma maior transparência do serviço contratado.

3.1 Virtualização

Podemos definir virtualização como o uma forma de se executar vários serviços, programas, ou até mesmo sistemas operativos num único equipamento físico. Esta técnica, traz uma série de vantagens como a gestão centralizada, ou seja, controlar grupos de computadores numa interface, permite a abstração de diferentes máquinas a nível de hardware e torna o processo de *backup* bastante

mais fácil graças a *Snapshots* dos estados das várias máquinas virtuais.

No entanto toda esta abstração proporciona um workload/overhead da coexistência de várias *Virtual Machines* (VM's) a partilharem o mesmo hardware, e consequentemente traz um maior nível de dificuldade de administração aliada a uma perda *performance*, fazendo também grande uso do espaço em disco.

4 Background

4.1 Infrastrucutre as a Service

Nos dias de hoje em que o poder computacional é necessário *on the fly* para disposição de serviços na web, surge um serviço que se dá pelo nome de IaaS, Infrastructure as a Service (IaaS). Este trata-se de uma infraestrutura informática que é aprovisionada e gerida através da Internet. No fundo são computadores/servidores ligados numa rede que estão disponíveis para criação de máquinas virtuais com um Sistema Operativo escolhido pelo utilizador, cabendo ao mesmo apenas o ónus de instalar o restante *software* que necessite.

Desta forma é possível escalar verticalmente, ou seja, otimizar os recursos da máquina virtual, e/ou escalar horizontalmente, isto é, aumentar o número de máquinas virtuais/CPUs para o cliente. Este conceito é conhecido como elasticidade e é uma das armas que torna a `Cloud` tão útil e apelativa.

Frequentemente existem casos quotidianos por exemplo de empresas como *Startups* com serviços na web que necessitam de poder computacional variável. Sendo assim existe uma necessidade de aumentar (escalar) ou diminuir o poder computacional consoante a necessidade. No entanto, existem certas situações em que nos vemos obrigados a implementar a nossa própria `Cloud`. Para tal existem ferramentas que transformam uma rede de servidores `Bare Metal` num `IaaS` como o `OpenStack` disponbilizando neste caso virtualização de hardware.

4.2 Orquestrador

O Orquestrador como ferramenta da `Cloud`, tem o papel complexo de coordenar, organizar e automatizar a entrega da infraestrutura (`IaaS`) ao serviço que o cliente está de momento a executar. Isto é, ao utilizarmos esta ferramenta estamos a dar o poder de decisão dos recursos bem como a faculdade da elasticidade sobre esses mesmos recursos ao orquestrador escolhido.

Esta ferramenta para além de eliminar um trabalho de administração controlada por humanos e mais suscetível a falhas, visa promover a redução de custos para o utilizador.
Supondo que temos dois serviços de `Cloud` como a Amazon Web Services (AWS) e `Google Cloud`, fazendo uso de um orquestrador com acesso a estas duas

clouds, será possível que esta ferramenta escolha qual a Cloud que irá fazer uso mediante a política escolhida no qual um dos parâmetros poderá abordar o custo.

Na nossa jornada apenas fez-se uso da API que liga o IaaS ao orquestrador proporcionada pelo mesmo, ignorando as suas valências de "orquestração inter-cloud"dado que usamos apenas uma Cloud privada.

4.3 Camada Aplicacional

Esta camada sendo o maior nível de abstração desta "pilha tecnológica", pretende facilitar a gestão de aplicações lançadas para a cloud de forma a que os programadores percam menos tempo na fase de *deploy* e possam focar-se mais na fase de desenvolvimento da aplicação.

Neste patamar procura-se facilitar ao cliente a definição de uma topologia de aplicação, e lançá-la para a cloud de forma simples.

Assim sendo, estaremos a responder a uma série de necessidades na indústria de *software*: reusabilidade, extensibilidade, flexibilidade, consistência e evolução.

Para responder a estas carências e para trabalharmos de forma mais abstraída utilizou-se o A4C em grande parte dos nossos cenários de experimentação.

4.4 Sobreposição de funcionalidades entre ferramentas

Este tipo de aplicações, dada a sua natureza, podem assumir mais do que um dos patamares definidos. Isto é, podem executar mais do que uma função.

O OpenStack sendo uma aplicação que proporciona uma gestão de recursos computacionais ligados em rede (Platform as a Service ou PaaS), proporcionando ao cliente um serviço de Infrastructure as a service ou IaaS, também pode funcionar como Orquestrador fazendo uso de um serviço como o Magnum que usa o plugin Heat que contém um Docker e Kubernetes permitindo a Orquestração.

A função principal do cloudify é a de orquestração, no entanto este também dispõe de uma *Graphical User Inteface* pelo que também executa uma camada aplicacional semelhante à do A4C.

Cenário	Componentes/Funcionalidade [IaaS ; Orquestrador ; App]
1	OpenStack Horizon; `Janus` 2.0 ; `A4C` 1.4.0
2	OpenStack Horizon; Cloudify 3 + Kubernetes 1.11 ; `A4C` 1.4.0
3	OpenStack Horizon; Cloudify 4 (Orquestrador e App)
4	OpenStack Horizon; Cloudify 4 ; `A4C` 2.0.0
5	Openstack Horizon; Cloudify 4 + Kubernetes 1.11 ; `A4C` 2.0.0

Tabela 1: Componentes mediante os cenários explorados

5 Componente Experimental

Para testarmos os vários orquestradores, construíu-se vários cenários com vários componentes mediante a funcionalidade que desempenha na nossa *stack* tecnológica.

5.1 Cenário 1

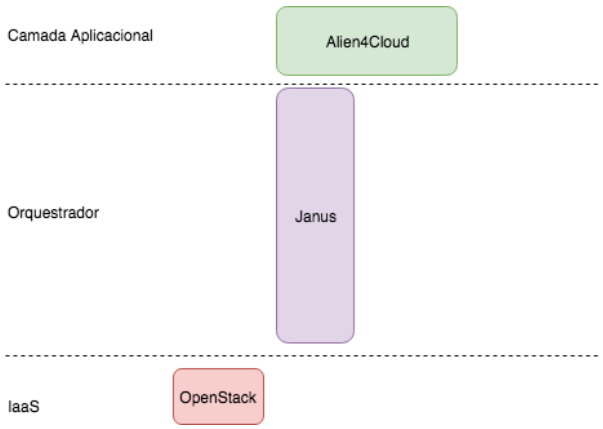

Figura 2: Imagem do cenário 1

Decidiu-se usar o `Janus` como primeira abordagem e como termo de comparação para os outros cenários.

Testou-se o OpenStack como IaaS, o `Janus` como Orquestrador e o `A4C` como App como é possível ver na figura 2. Começámos por criar uma máquina para o `Janus`, instalando todos os componentes necessários, entre eles o `Terraform` e o `Consul`, sendo estes componentes responsáveis pelo aprovisionamento e configuração de serviços em *runtime* dentro da infraestrutura respetivamente. Por fim, guardou-se a private key do keypair utilizado para ser possivel o aprovisio-

namento de software por SSH.

Posto isto, procedeu-se à integração do A4C no Janus, utilizando a versão 1.4.0, de forma a poder lançar máquinas para o Openstack, com topologias criadas no A4C com a da figura 3, orquestradas pelo Janus. Esta integração foi feita através da criação de uma nova máquina para o A4C, e de um plugin existente, que instalado na máquina do Janus, faz a ligação entre a camada aplicacional e o orquestrador.

Começou-se por satisfazer os requisitos mínimos do A4C, entre eles a instalação

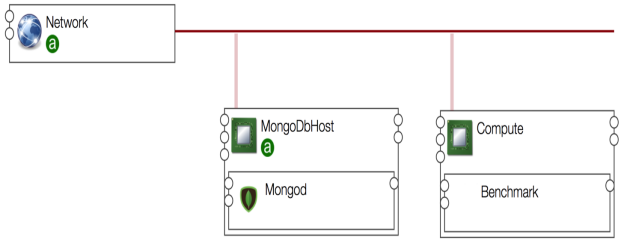

Figura 3: Topologia A4C

do JAVA 8 e o Python, e posteriormente inseriu-se o *plugin* na máquina do Janus, para ser configurada mais tarde.

Para configurar e testar esta integração, teve-se de inicializar um agente do Consul, e o servidor Janus na máquina, o que nos levou à primeira dificuldade. Deparamo-nos com um erro, que após analisado, nos fez supor que seria um problema de versionamento, nomeadamente do Consul e do Terraform. Procedeu-se então à identificação e instalação das versões corretas dos componentes, ou seja, aquelas que se complementavam com a versão 2.0 do Janus que estavamos a utilizar. Após tudo isto, o agente Consul e o servidor Janus foram iniciados com sucesso. Seguimos então para a máquina do A4C, onde iniciou-se o script existente para lançar o A4C, onde nos deparamos novamente com dificuldades, por problemas de versionamento associados ao Python.

Identificou-se o principal motivo para todas estas dificuldades como sendo a escassa/deficiente documentação do A4C, talvez por ser uma documentação OpenSource e com poucos contribuidores.

Com o A4C finalmente a funcionar, procedeu-se à configuração do plugin A4C Janus na UI que nos é oferecida pela aplicação. Para isso, na secção Administration/Plugins, inserimos o zip do plugin, na secção Administration/Orquestrators identificou-se o Janus como orquestrador e inseriu-se o IP correspondente à máquina do mesmo, por fim ativando o orquestrador. Por fim, na Secção Locations,

identificou-se a mesma como sendo o Openstack e configuraram-se os *resources* necessários, nomeadamente o `Janus.nodes.openstack. PublicNetwork` e o `Janus.nodes.openstack.Compute`, associando o nome da network previamente criada no Openstack ao resource PublicNetwork e todas as informações necessárias referentes às máquinas que iriam ser criadas através do `Janus`.

Com tudo isto devidamente configurado, procedeu-se ao *deploy* de uma instância `MongoDB`, em que a sua topologia pode ser facilmente encontrada nos repositórios GitHub do próprio `A4C`. Após alguns ajustes nos ficheiros de configuração da topologia, associados à diferença de versões, conseguiu-se obter uma máquina com uma base de dados `MongoDB` completametente funcional.

Como o *deploy* da instância `MongoDB` foi iniciada com sucesso, conclui-se que toda a stack funciona.

5.2 Cenário 2

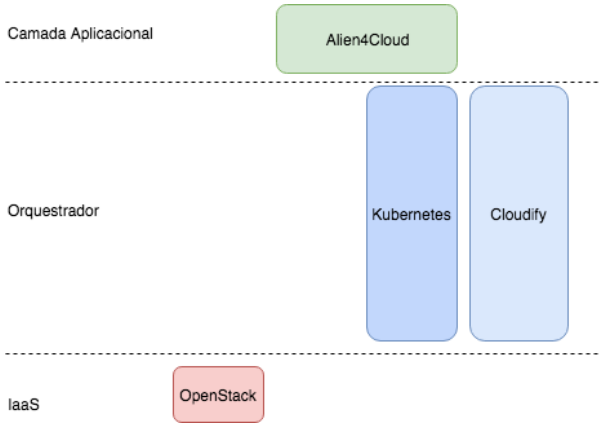

Figura 4: Imagem do cenário 2

Como abordagem seguinte, experimentou-se o `Kubernetes` como orquestrador como é descrito na figura 4. Este revelou-se um caso bastante particular na nossa jornada, sendo que esta ferramenta é orientada ao `Container` e não orientado à Virtual Machine (VM), isto é, lançando uma imagem de uma aplicação para uma estrutura virtual previamente criada, apelidada de `Kubernetes pod`, estaremos a criar um `Docker Container` no `kubernetes pod` com essa imagem instalada nesse mesmo `Container`.

A documentação encontrada nos sites oficiais revelou-se um pouco confusa, não pelo facto de ser escassa, mas pelo facto desta ferramenta poder ser lançada de várias formas consoante a necessidade de cada utilizador.

Foi lançado um `minikube` que consiste num `Cluster` ou conjunto de dois `containers` com apenas um `Master Node` e um `Pod Node` numa VM, sendo este um requisito mínimo para o Kubernetes funcionar. Ou seja, para que o Kubernetes possa lançar aplicações necessita de dois *containers*, sendo que um inicia instâncias, o `Master Node` e o outro executa-as, o `Pod Node`

Não se obteve grande dificuldade no que diz respeito à ligação ao `OpenStack` visto não ser necessário nada mais do que uma VM com um Sistema Operativo *Unix based* como o `Ubuntu`.

Uma das características desta ferramenta atualmente é que necessita estar acoplada a um orquestrador independente. A documentação que nos fez levar a este caminho foi a que fazia uso do *Cloudify Manager* da versão 3 do Orquestrador *Cloudify*, que foi facilmente ligada. No que diz respeito à ligação com a camada superior, o `A4C`, esta revelou-se mal-sucedida, devido à carência de plugins do Kubernetes para a versão `1.4.0` do `A4C`.

Não obstante deste percalço, foi possível fazer uso da restante *stack* tecnológica abaixo da camada aplicacional. Sendo que o processo de lançamento de aplicações através de imagens `Docker` seriam despoletadas pelo `Master Node` na ferramenta CLI `kubectl` que por sua vez irá criar um `Docker container` com a respetiva imagem instalada num dos *pods* do *Kubernetes Cluster* (no nosso caso particular, apenas tínhamos um *Pod* disponível).

Desta forma, não nos foi possível proceder ao lançamento de uma topologia através da camada aplicacional, isto é, no `A4C`, pelo conclui-se que em relação ao *cenário 1* esta solução não é viável, não funciona como expectado.

5.3 Cenário 3

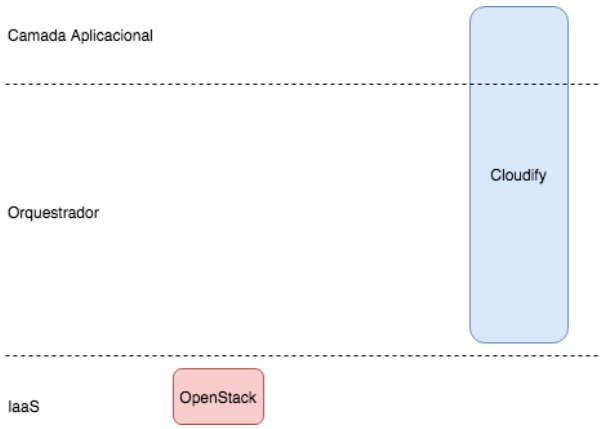

Figura 5: Imagem do cenário 3

Após estas abordagens, experimentou-se desta vez o Cloudify 4, como se vê na figura 5 . Sendo que este tem as ferramentas necessárias para fazer o trabalho de mais do que uma camada, utilizou-se tanto como orquestrador como na camada aplicacional.

Deparamo-nos também aqui com alguns problemas de versionamento, nomeadamente as versões do Cloudify. Utilizou-se numa primeira tentativa a versão Community do Cloudify.

Verificámos que a documentação apresentada no site relativa a esta ferramenta tinha muitas diferenças comparando com a versão Community, referindo configurações e secções da UI que não existiam nesta versão.

Percebeu-se então que talvez a documentação fosse feita para a versão Enterprise, e por essa razão passou-se à segunda tentativa, utilizando a versão Enterprise. Nesta versão, foi possível lançar uma máquina com um `manager` do `Cloudify`, devidamente configurado. Esta configuração, seguindo a documentação disponibilizada, é bastante fácil de conseguir.

Em seguida, fez-se upload de um `blueprint` exemplo, topologia de uma aplicação do Cloudify, para deploy de uma topologia MongoDB, que pode ser encontrada fácilmente no GitHub do próprio Cloudify.

Ao tentar fazer este deploy, deparamo-nos com a necessidade de uma configuração extra, pois este deploy exigia várias informações para se completar, nomeadamente sobre o nosso Openstack (authentication url, private key, public key, etc). Posto isto, verificaram-se todas as informações que seriam necessárias sendo as mesmas adicionadas como mecanismo de store de informação sensível (secrets), uma funcionalidade do Cloudify que permite guardar informações sensíveis como variáveis, de forma a não ser necessário o seu input de cada vez que se faz um deploy.

Verificou-se também que para o deploy de alguma topologia, seria necessário um primeiro deploy de uma topologia de uma `network Openstack`, em que usou-se mais uma vez um `blueprint` exemplo para este efeito.

Posto isto, tentámos novamente o deploy da topologia `mongoDB`, e mais uma vez foram encontradas dificuldades, dado que estavam em falta *plugins* necessários para a instalação, *plugins* esses que são fácilmente encontrados no website do `Cloudify` e também fácilmente instalados através da UI disponibilizada.

Associada a esta "tentativa-erro"dos `plugins`, e ao facto de que este *deploy* utiliza a network acima descrita, encontrámos uma dificuldade ainda maior pois, a cada tentativa de *deploy*, eram criadas networks, routers e security groups no `Openstack`, que não poderiam ser apagados sem acesso como administrador, ocupando os recursos disponíveis e impossibilitando novas tentativas de deploy.

Após superar esta dificuldade, conseguimos então um deploy da topologia MongoDB, ainda que parcial devido a limitações em termos de recursos, logo é de concluir que a stack tecnológica funciona.

5.4 Cenário 4

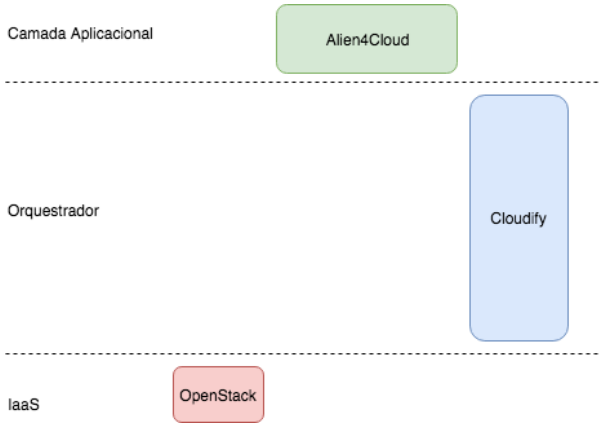

Figura 6: Imagem do cenário 4

Depois de conseguido, ainda que de maneira parcial, o deploy da instância MongoDB através do Cloudify, decidiu-se substituir o Cloudify na camada aplicacional e usar, desta vez, o `A4C` 2.0.0, tal como pretende demosnstrar a figura 6.

Começou-se por instalar o plugin do Cloudify 4 na nova máquina `A4C`, da mesma forma que foi instalada a anterior versão. A configuração é também bastante semelhante à das versões anteriores, apenas com uma funcionalidade adicional de segurança, ainda que opcional. Não foi experimentada esta última por entendermos que não era realmente relevante para o resultado do trabalho.

Depois de tudo devidamente instalado e configurado, utilizou-se então a mesma topologia `MongoDB` anterior, com apenas algumas alterações de maneira a satisfazer o facto de estarmos a usar um orquestrador diferente do anterior. Ao tentar o *deploy*, constatamos que havia conexão entre o `A4C` e o `Cloudify`, pois na UI deste último aparecia a tentativa de *deploy* da dita topologia. Ainda assim, através dos *logs* do `Cloudify`, vimos que o deploy não tinha sido bem sucedido, com um erro associado ao Python, mas demasiado geral para dar alguma pista sobre a possível causa da falha. Apesar das poucas certezas dada a vaguidão do erro, supomos que seja alguma falta de dependências, que não estarão contempladas nas documentações do `A4C` e `Cloudify`, pelo que concluímos que em comparação com os cenários menicionados acima, esta stack tecnológica não funciona devido a erros devido ás dependências referidas acima.

5.5 Cenário 5

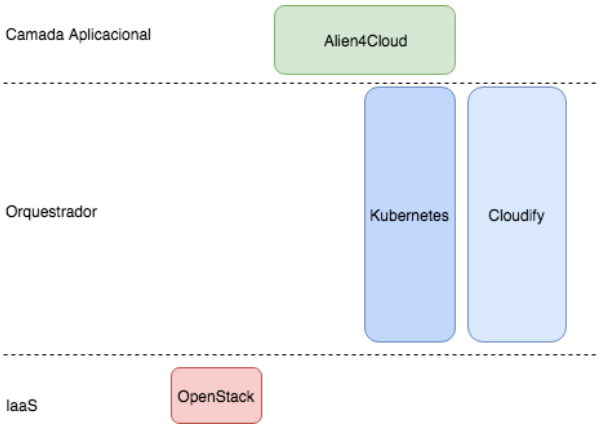

Figura 7: Imagem do cenário 5

Como última abordagem, com a vantagem de ter já o Cloudify 4 devidamente configurado, tentou-se utilizar novamente o Kubernetes como orquestrador, como mostra a figura 7. Desta vez, estando a ser usadas outras versões dos componentes, conseguimos encontrar os plugins necessários para que esta integração fosse possível.

Nesta abordagem, a instalação do Kubernetes foi feita através de um blueprint disponibilizado pelo Cloudify, em que apenas foi necessário adicionar alguns `secrets` com informações que seriam utilizadas pelo deploy. Esta abordagem acabou por ser uma tentativa incompleta, pois a exigência de recursos para a instalação dos módulos do Kubernetes impediu-nos de avançar, dada a limitação de recursos que temos no Openstack.

Apesar disto, conseguiu-se facilmente supor que o *deploy* iria ser bem sucedido, dado que alguns dos módulos ficaram devidamente instalados antes dos recursos chegarem ao seu limite. Dado que a conexão com o `A4C` neste cenário seria feita novamente através do `Cloudify`, supomos também que seria bem sucedido, ainda que com a possibilidade de existir o mesmo erro com que nos deparamos no cenário acima, pelo que concluímos que a stack tecnológica funciona.

6 Taxonomia de aplicações para orquestração

Como resultado desta experimentação, analisámos alguns pontos sobre os quais achamos importante debater entre os vários cenários.

Ferramenta	OS	#CPU	RAM	Storage
`Janus`	Linux x86_64	2	2GB	-
Cloudify 3	Linux x86_64/Windows/OS X	2	4GB	5GB
Cloudify 4	Linux x86_64/Windows/OS X	2	4GB	5GB
Kubernetes *	CentOS 7	2	4GB	-
`A4C`	Linux x86_64/OS X	2	2GB	-

Tabela 2: Requisitos das várias ferramentas exploradas

6.1 Integração

Começamos por analisar a integração entre a camada IaaS e a camada de orquestração e entre esta última e a camada aplicacional. De um modo geral, a integração entre as camadas é bastante fácil, pois para todas as ligações existem já plugins disponíveis.

Após instalados, estes tratam de toda a conexão entre as APIs, apenas necessitando de pequenas configurações, nomeadamente a introdução do IP das máquinas onde se encontram as ferramentas de cada camada. Uma das grandes perguntas acerca deste trabalho seria se realmente as APIs se ligariam tão bem como é dito pelas ferramentas envolvidas.

Numa primeira análise, seguindo a documentação disponibilizada, tivemos algumas dúvidas quanto a esta facilidade de integração, isto é, apercebemo-nos da possibilidade de poder haver, por exemplo, alguns métodos que não estivessem implementados. Ainda assim, em nenhum dos cenários tivemos problemas com a integração através dos plugins, não obtendo qualquer erro dos esperados anteriormente.

6.2 Versionamento

6.3 Complexidade do deploy

Analisámos também esta experimentação ao nível de infraestrutura, isto é, das máquinas virtuais, no deploy das ferramentas estudadas. Esta tabela contempla os requisitos mínimos da máquina para cada ferramenta utilizada.

Dada a complexidade dos vários cenários, dos requisitos de infraestrutura, erros e dificuldades encontradas nesta experimentação, decidiu-se elaborar uma tabela que mostra a dificuldade de instalação, configuração e deploy de instâncias de possíveis *apps* por parte do cliente utilizando estas mesmas ferramentas.

Esta tabela mostra os vários cenários e uma classificação em termos de dificuldade baseada na nossa opinião, que é quantificada de **1 a 5**, sendo o **1** o mais fácil e o **5** o mais difícil.

Cenário	Compontentes/Funcionalidade	Dificuldade
1	OpenStack Horizon; `Janus`; `A4C` 1.4.0	2
2	OpenStack Horizon; Cloudify 3 + Kubernetes; `A4C` 1.4.0	4
3	OpenStack Horizon; Cloudify 4 (Orquestrador e App)	3
4	OpenStack Horizon; Cloudify 4; `A4C` 2.0.0	3
5	Openstack Horizon; Cloudify 4 + Kubernetes; `A4C` 2.0.0	2

Tabela 3: Dificuldade de *deploy* dos vários cenários

6.4 Recomendação

Depois do estudo de todos estes cenários e pontos de análise, chegou-se a algumas conclusões que pretendemos expor em tom de recomendação ao utilizador/leitor .

No caso do utilizador dispor de recursos computacionais limitados, isto é, poucos CPU's e memória RAM na Cloud, o *cenário 1* será o mais indicado dada baixa necessidade destes mesmos recursos como se pode verificar observando a tabela 6.3.

No caso de não haver este tipo de não haver tanta limitação a nível de infraestrutura como no caso anterior, o melhor cenário será o **cenário 4**. Cada camada está bem definida em termos de ferramentas, sendo que cada ferramenta tem apenas uma função específica. Neste cenário o utilizador beneficia ainda de um maior controlo sobre o deploy, dado que a `UI` do `Cloudify` oferece um `log` dos erros. Beneficia também do `A4C` pela sua facilidade de estruturar a topologia da aplicação requerida pelo utlizador.

No caso da intenção do utilizador ser um deploy de apps em containers e fazer uso de imagens disponíveis numa comunidade como o *Docker Hub* a melhor opção seria o **cenário 5**. Neste cenário o utilizador beneficia também do maior controlo falado anteriormente, associado a usar o Cloudify como orquestrador principal.

7 Conclusão

De um modo geral, foi feito um trabalho profundo no que diz respeito à exploração de orquestradores na Cloud tendo em vista a recomendação mediante a necessidade do utilizador.

É de frisar que nem sempre obtivemos o resultado expectável no entanto este não foi visto como um percalço mas sim como uma forma de avaliação em termos qualitativos da solução implementada.

Podemos dizer que este tipo de tecnologia é bastante importante nos dias de hoje mas necessita de um conhecimento e/ou *background* mais aprofundado por parte do utilizador.

8 Bibliografia

- Kubernetes Documentation
- Medium Blog - Tutorial Kubernetes + OpenStack
- Deploy a hybrid nodecellar application on Kubnernetes through A4C/Cloudify Tutorial
- Documentação Cloudify 4
- A4C - Cloudify4Driver